

SIX WEEKS SUMMER TRAINING REPORT

On

**JAVA AND DATA STRUCTURES AND ALGORITHM**

Submitted by

**SHREYASH PANDEY**

**Registration No: 12110805**

**Program Name: B.Tech.(CSE)**

Under the Guidance of

**Mr. Sehajpreet Singh**

**Java and DSA mentor**

# School of Computer Science and Engineering

# Lovely Professional University. Phagwara

# (June- July 2023)

**DECLARATION**

I hereby declare that I have completed my six weeks summer training at **CipherSchools** from **6th June 2023** to **17th July 2023** under the guidance of **Mr. Sehajpreet Singh**. I declare that I have worked with full dedication during these six weeks of training and my learning outcomes fulfill the requirements of training for the award of degree of **B.Tech.(Computer Science & Engineering)** at Lovely Professional University, Phagwara.

Shreyash Pandey

Registration No: 12110805

## **ACKNOWLEDGEMENT**

To acknowledge all the persons who had helped in completing the training and project is not possible for any trainee. However, despite all that, it becomes a foremost responsibility of the trainee and the part of research ethics to acknowledge those who had played a significant role in the completion of the training and project.

So, in the same sequence at very first, we would like to acknowledge **Mr. Sehajpreet Singh**, the mentor of our Training. He has been instrumental in guiding and helping us while undergoing the planning phase of our project and helped clear the concepts related to the topics and project. Because the **Training and Placement Cell of** **School of Computer Science & Engineering** has allowed me to do the summer training, I would like to thank for the same. I would like to thank CipherSchools also for organizing such a wonderful summer training program.

Later, I would like to confer the flower of acknowledgment to my parents because of whom we got the existence in the world for the inception and the conception of this training and project. Rest all those people who helped us are not only a matter of acknowledgment but also authorized to share our success.

With Regards

Shreyash Pandey

**CERTIFICATE**

A certificate of completion with orange and white text

Description automatically generated

**TABLE OF CONTENTS**

Inner first page…………………………………………………………………….(i)

Declaration…………………………………………………………………….......(ii)

Acknowledgement………………………………………………………………...(iii)

Certificate………………………………………………………………………….(iv)

Table of Contents…………………………………………………………………...(v)

1. Introduction…………………………………………………………………….........6
2. Technology Learnt…………………………………………………………………..8
   1. Java and it’s basics……………………………………………………...……...8
   2. Data Structures and Algorithms……………………………………………....13
3. Reason for choosing this technology……………………………………………….30
4. Profile of the Problem………………………………………………………………32
5. Problem Analysis……………………………………………………………………33
   1. Product Definition……………………………………………………………..33
   2. Feasibility Analysis……………………………………………………………33
6. Implementation………………………………………………………………………34
7. Learning Outcome for Training……………………………………………………...38
8. Gantt Chart…………………………………………………………………………..40
9. Project Legacy………………………………………………………………………..41
   1. Technical Lesson learnt………………………………………………………....41

10. References……………………………………………………………………...………42

1. **INTRODUCTION**

Welcome to the introductory page of my training report, documenting the comprehensive and enriching journey I undertook with CipherSchools in their meticulously designed program - Java and Data Structures & Algorithms (DSA). This report encapsulates my experiences, insights, and growth throughout the duration of this transformative training.

* **CipherSchools:**

CipherSchools, a renowned institution synonymous with excellence in technical education, has been a guiding light in my quest for knowledge and skill enhancement. Their commitment to fostering proficiency in software development and computer science is reflected in their well-structured curriculum and exceptional faculty. The Java and DSA training program, which I had the privilege of attending, promised to be a voyage of discovery and enhancement of my programming prowess.

* **Java Proficiency:**

The training embarked with an immersive dive into Java programming. Guided by experienced mentors, I delved into the nuances of the language, honing my skills in object-oriented programming, multi-threading, exception handling, and more. The comprehensive lectures, practical coding sessions, and real-world examples have empowered me to develop robust and efficient Java applications.

* **Learning Data Structures & Algorithms:**

As the training progressed, the spotlight shifted towards mastering Data Structures and Algorithms (DSA). I was introduced to the fundamental building blocks of efficient software development. Through hands-on exercises, coding challenges, and interactive sessions, I acquired the ability to analyze, design, and implement complex algorithms. The newfound proficiency in DSA not only elevated my problem-solving skills but also equipped me to tackle coding interviews and challenges with confidence.

* **Experiential Learning:**

One of the most invaluable aspects of the training was the emphasis on experiential learning. Collaborative projects, coding competitions, and group discussions allowed me to apply theoretical knowledge in real-world scenarios. These experiences fostered a deeper understanding of the concepts and instilled a sense of teamwork, adaptability, and innovation.

* **Personal Growth:**

Beyond the technical know-how, this journey was a period of personal growth and self-discovery. The challenges I faced and overcame, the friendships I forged, and the mentors I connected with, all contributed to my holistic development. The training instilled in me a sense of discipline, perseverance, and continuous learning - qualities that are essential not just in the world of programming, but in every facet of life.

* **Gratitude:**

As I conclude this introductory segment, I extend my heartfelt gratitude to CipherSchools for providing me with this unparalleled opportunity for learning and growth. The Java and DSA training has laid a strong foundation upon which I aim to build a thriving career in software development. The knowledge gained and experiences cherished during this training will forever shape my professional journey, driving me to excel and innovate in the ever-evolving landscape of technology.

1. **TECHNOLOGY LEARNT**

During the 6-week online summer training “Java and Data Structures & Algorithms Program”, I have learnt basics of Java and Data Structures and applied those concepts in solving some standard competitive coding problems using Java. The brief outlines of the contents covered is as following:

* 1. **Java and its Concept**: Java is a programming language and a platform. Java is a high level, robust, object-oriented, and secure programming language. Various concepts learned in Java

Are as follows:

* **Features of Java**



**Figure 2.1 Features of Java**

1. **Object-Oriented**

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

Object and Class

Encapsulation

Inheritance

Polymorphism and Abstraction

1. **Platform Independent**

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).



**Figure 2.2 Platform Independency**

1. **Secured:**

Java is best known for its security. With Java, we can develop virus-free systems.



**Figure 2.3 Way to achieve security.**

* **Objects and Classes:**

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

1. //Java Program to illustrate how to define a class and fields
2. //Defining a Student class.
3. **class** Student{
4. **int** id;//field or data member or instance variable
5. String name;
6. **public** **static** **void** main(String args[]){
7. //Creating an object or instance
8. Student s1=**new** Student();//creating an object of Student
9. //Printing values of the object
10. System.out.println(s1.id);//accessing member through reference variable
11. }
12. }

* **Inheritance:**

The process of obtaining the data members and methods from one class to another class is known as inheritance. The process of obtaining the data members and methods from one class to another class is known as inheritance. It is one of the fundamental features of object-oriented programming.

**Why use Inheritance?**

* For Method Overriding (used for Runtime Polymorphism).
* Its main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class.
* For code Re-usability

**Figure 2.4 Types of Inheritance**

A diagram of a class

Description automatically generated

* **Enacapsulation:**

Encapsulation is a process of wrapping data and methods in a single unit called encapsulation. Encapsulation is achieved in java language by class concept. Combining of state and behavior in a single container is known as encapsulation. In java language encapsulation can be achieve using class keyword, state represents declaration of variables on attributes and behavior represents operations in terms of method.

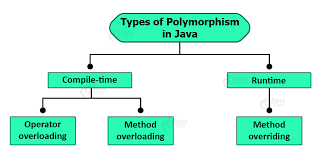
A diagram of a pill

Description automatically generated

**Figure 2.5 Encapsulation**

* **Polymorphism:**

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word **“poly”** means many and **“morphs”** means forms, so it means many forms.



**Figure 2.6 Polymorphism**

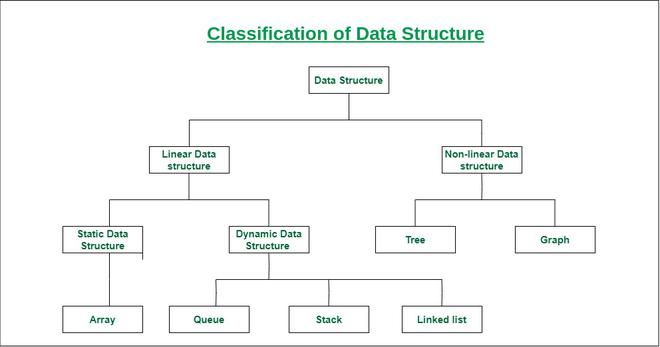
**Types of Java polymorphism**

In Java polymorphism is mainly divided into two types:

1. Compile-time Polymorphism
2. Runtime Polymorphism

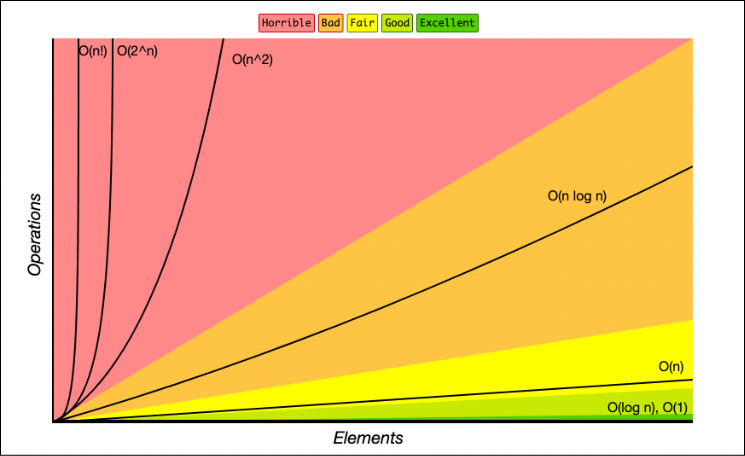
**2.2 Data Structures and Algorithms:** It's a fundamental concept in computer science that deals with organizing and manipulating data efficiently and designing algorithms to solve computational problems. Data structures are ways of organizing and storing data in a way that enables efficient access and modification.

* **Classification of Data Structures:**



**Figure 2.7 Data Structures Classification**

**Time Complexity:** Time complexity is a concept used in computer science to measure the efficiency of an algorithm in terms of the time it takes to complete based on the input size



**Figure 2.8 Big O Complexity Chart**

* **Sorting Techniques:** Sorting algorithms are used to arrange the elements of a data structure in a particular order such as ascending or descending order**.** Various Sorting techniques are as follows.

1. **Bubble Sort**
2. **Insertion Sort**
3. **Selection Sort**
4. **Merge Sort**
5. **Quick Sort**

public class Quick\_Sort

{

public static void quickSort(int arr[],int start,int last)

{

if(last>start)

{

int q=partion(arr,start,last);

quickSort(arr,start,q-1);

quickSort(arr,q+1,last);

}

}

public static void swap(int arr[],int a,int b)

{

int temp=arr[a];

arr[a]=arr[b];

arr[b]=temp;

}

public static int partion(int arr[],int s,int l)

{

int i=s-1;

int j=s;

int pivot=l;

while(j<l)

{

if(arr[j]<arr[pivot])

{

i++;

swap(arr,i,j);

}

j++;

}

swap(arr,i+1,pivot); e swapping here

return i+1;

}

**Code 2.1 Quick-Sort**

**Table 2.1: Comparison of Complexities of different Sorting Algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Algorithm | Best Case | Average Case | Worst Case |
| Bubble Sort | n2 | n2 | n2 |
| Selection Sort | n2 | n2 | n2 |
| Insertion Sort | n | n2 | n2 |
| Quick Sort | n log2 n | n log2 n | n2 |
| Merge Sort | n log2 n | n log2 n | n log2 n |

* **Array:** An array is a linear data structure that stores a collection of elements of the same data type, each identified by its index or position.

**Operations performed on array:**

1. **Insertion:**
2. **Deletion**
3. **Traversal**
4. **Merging**
5. **Sorting**

**Problem Solved:**

* **Subarray with given sum:** Given an unsorted array A of size N that contains only positive integers, find a continuous sub-array that adds to a given number S and return the left and right index (1-based indexing) of that subarray. In case of multiple subarrays, return the subarray indexes which come first on moving from left to right.

**Note: -** You have to return an Array List consisting of two elements left and right. In case no such subarray exists return an array consisting of element -1.

**Example 1:**

**Input:**

N = 5, S = 12

A [] = {1,2,3,7,5}

**Output: 2 4**

**Explanation:** The sum of elements from 2nd position to 4th position is 12

**Code 2.2 Subarray with Given Sum**

class Solution

{ //Function to find a continuous sub-array which adds up to a given number.

static ArrayList<Integer> subarraySum(int[] arr, int n, int s)

{

ArrayList<Integer> a= new ArrayList<>();

int start=0,sum=0,have=-1;

for(int i=0;i<n;i++)

{

sum=sum+arr[i];

if(sum>s)

{

while(sum>s)

{

sum=sum-arr[start++];

}

}

if(sum==s && start<=i)

{

have++;

a.add(start+1);

a.add(i+1);

break;

}

}

if(have==-1)

{

a.add(have);

}

return a;

}

}

* **Reverse array in groups**
* **Spirally traversing matrix**
* **Sort arrays of 0s,1s and 2s**
* **Implement two stacks in one array.**
* **First element to occur k times**
* **Smaller on left.**
* **Adding ones**
* **LinkedList**: A linked list is a linear data structure where elements (nodes) are connected through pointers, allowing dynamic insertion and deletion of elements. Types of LinkedList are

1. **Singly LinkedList**
2. **Doubly LinkedList**
3. **Circular LinkedList**

**Operations performed on LinkedList:**

1. **Insertion:**
2. **Deletion**
3. **Traversal**
4. **Merging**
5. **Sorting**

**Problem Solved:**

* **Detect loop in LinkedList.**
* **Nth node from end of LinkedList**
* **Finding Middle element of LinkedList**
* **Reverse a LinkedList**
* **Reverse LinkedList in group**
* **Union of two LinkedList**
* **Remove loop in Linked List:** Given a linked list of N nodes such that it may contain a loop. A

loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.Remove the loop from the linked list, if it is present, i.e., unlink the last node which is forming the loop.

**Example 1:**

**Input:**

N = 3

value [] = {1,3,4}

X = 2

**Output:** 1

**Code 2.3 Remove Loop in a LinkedList**

class Solution

{

//Function to remove a loop in the linked list.

public static void removeLoop(Node head)

{

Node slow=head;

Node fast=head;

while(fast!=null && fast.next!=null)

{

slow=slow.next;

fast=fast.next.next;

if(fast==slow)break;

}

if(slow==head)

{

while(fast.next!=head)

{

fast=fast.next;

}

fast.next=null;

}

if(slow==fast)

{

slow=head;

while(slow.next!=fast.next)

{

slow=slow.next;

fast=fast.next;

}

fast.next=null;

}

}

}

* **Stack:** A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle, where

elements are added and removed from the top, allowing efficient access and manipulation of the most recently added items.

**Stack, as an abstract data type (ADT), supports the following operations:**

1. **Push:** Adds an element to the top of the stack.
2. **Pop:** Removes and returns the top element from the stack.
3. **Peek ():** Returns the top element without removing it.
4. **isEmpty() :** Checks if the stack is empty.
5. **Size():** Returns the number of elements in the stack.

**Problem Solved**

* **Implement two stacks in one array**
* **Stack using two queues.**
* **Implement Stack using two LinkedList**
* **Reverse a String using Stack**
* **Queue Using Stack**

class Queues

{

Queue<Integer> q1 = new LinkedList<Integer>();

//Function to push an element into stack using two queues.

void push(int a)

{

int k=0;

q1.add(a);

while(k!=q1.size()-1)

{

q1.add(q1.poll());

k++;

}

}

//Function to pop an element from stack using two queues.

int pop()

{

if(q1.isEmpty())return-1;

else

return q1.poll();

}

}

**Code 2.4: Stack using two queues**

**Code 2.5: Queue using two stacks.**

class Queue

{

Stack<Integer> s1 = new Stack<Integer>();

Stack<Integer> s2 = new Stack<Integer>();

/\*The method pop which return the element poped out of the stack\*/

int dequeue()

{

if(s1.isEmpty())return -1;

else

{

while(!s1.isEmpty())

{

s2.push(s1.pop());

}

int x=s2.pop();

while(!s2.isEmpty())

{

s1.push(s2.pop());

}

return x;

}

}

/\* The method push to push element into the stack \*/

void enqueue(int x)

{

s1.push(x);

}

}

* **Queue:** A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle, where elements are inserted at the rear and removed from the front**.**

**Queue, as an abstract data type (ADT), supports the following operations:**

* **add ():** Add an element to the back of the queue.
* **poll ():** Remove and return the element from the front of the queue.
* **Peek ():** Retrieve the element at the front of the queue without removing it.
* **IsEmpty ():** Check if the queue is empty.
* **Size ():** Get the number of elements currently in the queue.
* **clear():** Remove all elements from the queue, making it empty.

**Problem Solved**

* **Maximum Diamonds**
* **Stack using Queue.**

// -------------------------------Approach:1 making pop () costly----------------------------------------

class Queues

{

Queue<Integer> q1 = new LinkedList<Integer>();

Queue<Integer> q2 = new LinkedList<Integer>();

//Function to push an element into stack using two queues.

void push(int a)

{

q1.add(a);

}

//Function to pop an element from stack using two queues.

int pop()

{

if(q1.isEmpty())return -1;

else

{

while(q1.size()!=1)

{

q2.add(q1.poll());

}

}

while(!q2.isEmpty())

{

q1.add(q2.poll());

}

return q1.poll();

}

}

// ---------------------------------------------------Approach 2 Making push costly---------------------------------------------------

class Queues

{

Queue<Integer> q1 = new LinkedList<Integer>();

Queue<Integer> q2 = new LinkedList<Integer>();

//Function to push an element into stack using two queues.

void push(int a)

{

while(!q1.isEmpty())

{

q2.add(q1.poll());

}

q1.add(a);

while(!q2.isEmpty())

{

q1.add(q2.poll());

}

}

//Function to pop an element from stack using two queues.

int pop()

{

if(q1.isEmpty())return -1;

else

return q1.poll();

}

}

// -----------------------------------------------------Approach 3 Stack using single queue----------------------------------------------

class Queues

{

Queue<Integer> q1 = new LinkedList<Integer>();

//Function to push an element into stack using two queues.

void push(int a)

{

int k=0;

q1.add(a);

while(k!=q1.size()-1)

{

q1.add(q1.poll());

k++;

}

}

//Function to pop an element from stack using two queues.

int pop()

{

if(q1.isEmpty())return-1;

else

return q1.poll();

}

}

**Code 2.6: Making pop() Costly**

**Code 2.7: Making push () Costly**

// ---------------------------------------------------Approach 2 Making push costly-------------------------------

class Queues

{

Queue<Integer> q1 = new LinkedList<Integer>();

Queue<Integer> q2 = new LinkedList<Integer>();

//Function to push an element into stack using two queues.

void push(int a)

{

while(!q1.isEmpty())

{

q2.add(q1.poll());

}

q1.add(a);

while(!q2.isEmpty())

{

q1.add(q2.poll());

}

}

//Function to pop an element from stack using two queues.

int pop()

{

if(q1.isEmpty())return -1;

else

return q1.poll();

}

}

**Code 2.8: Using Single Queue**

class Queues

{

Queue<Integer> q1 = new LinkedList<Integer>();

//Function to push an element into stack using two queues.

void push(int a)

{

int k=0;

q1.add(a);

while(k!=q1.size()-1)

{

q1.add(q1.poll());

k++;

}

}

//Function to pop an element from stack using two queues.

int pop()

{

if(q1.isEmpty())return-1;

else

return q1.poll();

}

}

* **Tree:** A tree is a hierarchical data structure that consists of nodes connected by edges. Each tree

has a root node as its starting point, and each node can have zero or more child nodes, forming a branching structure. Nodes in a tree are typically organized in levels or layers, with the root node at level 0 and its immediate children at level 1, and so on.

**Problem Solved:**

* **Diameter of a Binary Tree**
* **Height of Binary Tree**
* **Determine if Two Trees are Identical.**
* **Level order traversal in spiral form**
* **Lowest Common Ancestor in a BST**
* **Lowest Common Ancestor in a Binary Tree**
* **Two Mirror Trees**

class Solution

{

//Function to check if two trees are identical.

boolean isIdentical(Node root1, Node root2)

{

if(root1==null && root2==null)return true;

if(root1 ==null || root2==null)return false;

return root1.data==root2.data && isIdentical(root1.left,root2.left) && isIdentical(root1.right ,root2.right);

}

}

**Code 2.9: Check if tree is mirror or not.**

class Solution

{

//Function to return the lowest common ancestor in a Binary Tree.

Node lca(Node root, int n1,int n2)

{

if(root==null)return null;

if(root.data==n1 || root.data==n2)return root;

Node Lca=lca(root.left,n1,n2);

Node Rca=lca(root.right,n1,n2);

if(Lca!=null && Rca!=null)return root;

else if(Lca!=null && Rca==null)return Lca;

else if(Lca==null && Rca!=null)return Rca;

else

return null;

}

}

**Code 2.10: Lowest Common ancestor of binary tree**

**Code 2.11: Level order traversal in spiral form**

class Spiral

{

//Function to return a list containing the level order

//traversal in spiral form.

ArrayList<Integer> findSpiral(Node root)

{

ArrayList<Integer> s=new ArrayList<Integer>();

Stack <Node> a=new Stack<>();

Stack <Node> b= new Stack<>();

if(root==null)s.add(-1);

a.add(root);

while(!a.isEmpty() || !b.isEmpty())

{

while(!a.isEmpty())

{

Node x=a.pop();

s.add(x.data);

if(x.right!=null)

b.add(x.right);

if(x.left!=null)

b.add(x.left);

}

while(!b.isEmpty())

{

Node y=b.pop();

s.add(y.data);

if(y.left!=null)

a.add(y.left);

if(y.right!=null)

a.add(y.right);

}

}

return s;

}

}

* **Graphs**: A graph is a data structure that consists of a set of nodes (vertices) and a set of edges

that connect pairs of nodes. Graphs are used to represent relationships between various entities and can model a wide range of real-world scenarios, such as social networks, transportation systems, computer networks, and more.

**Graph representation:**

* **Adjacency List**

class graph

{

Map<Integer,Set<Integer>> a=new HashMap<>();

void addvertex(int b)

{

if(a.containsKey(b))

{

System.out.println("ERROR: The vertex "+b+" Already exist with neighbours: b--> "+a.get(b));

return;

}

a.put(b,new HashSet<>());//in new vertex we are adding the new Set

}

void addEdge(int start,int end)

{

a.get(start).add(end); //here we are getting key of map then at that key we are adding data //

//means that key is the name ot the SET in which we are adding the data

}

void print()

{

for(int x:a.keySet())

{

System.out.print("Vertex "+x+" ");

System.out.println(a.get(x));//getting the value at the x key of the map

}

}

}

**Code 2.12: Adjacency List representation using Hash and Set**

**Code 2.13: Adjacency List representation using Array**

class graph

{

//int arr[]=new int[];

int Vertices;

//or int v;

LinkedList<Integer> arr[];// =new LinkedList[size];

graph(int v)

{

Vertices=v;

//then this.v=v;

arr= new LinkedList[v];

for(int i=0;i<v;i++)

{

arr[i]=new LinkedList<>();

}

}

void addEdge(int start,int end)

{

arr[start].add(end);// adding at start vertex it's neighbour here start is source and end is destination

//arr[end].add(start); if our grap is bidirectional

}

void print()

{

for(int i=0;i<Vertices;i++)

{

System.out.print("Node "+i+" Neighbours are: ");

for(int x:arr[i])

{

System.out.print(x+" ");

}

System.out.println();

}

}

}

**Problem Solved:**

* **BFS**
* **DFS**
* **Set:** A set data structure is a collection of unique elements with no particular order. It offers methods to add, remove, and check for element presence, making it valuable for tasks that require unique item storage.

public class lecture52

{

public static void main(String args[])

{

Set<Integer> s=new HashSet<>();

// if we impelement it with the LinkedHasSet then it is ordered Set but even then we can not access it through index

System.out.println(s.isEmpty());

s.add(20);

s.add(50);

s.add(45);

for(int a:s)System.out.print(a+" ");

System.out.println();

System.out.println(s.isEmpty());

System.out.println(s.contains(45));

System.out.println("size "+s.size());

// trying to add duplicate elements in the set

s.add(45);

//now let see if elements added then size will automatically increase

System.out.println("size "+s.size());

s.remove(45);

System.out.println("size "+s.size());

System.out.println(s.contains(45));

System.out.println(s.isEmpty());

s.clear();

System.out.println(s.isEmpty());

}

}

**Code 2.14: Implementation of Set**

* **Map:** A map data structure in Java is a collection that stores key-value pairs, where each key is

unique and maps to a specific value. The Map interface provides methods to add, retrieve, update, and remove elements based on their keys. Some common implementations of the Map interface in Java include HashMap, TreeMap, LinkedHashMap, and Hashtable, each with its own characteristics and performance trade-offs. Maps are widely used for tasks like indexing, caching, and organizing data relationships in Java applications.

import java.util.\*;

public class lecture52MAP

{

public static void main(String args[])

{

Map<String,Integer> a=new HashMap<>();

a.put("1st",20);

a.put("2nd",10);

a.put("3rd",2);

a.put("4th",98);

a.put("5th",76);

System.out.println(a.get("3rd"));

System.out.println(a.get("4th"));

a.remove("4th");

System.out.println(a.get("4th"));

System.out.println(a.keySet());

System.out.println(a.entrySet());

System.out.println(a.containsKey("8th"));

}

}

**Code 2.15: Implementation of Map**

1. **Reason for Choosing Java and (DSA) as Preferred Technologies**

In the realm of software development, the choice of programming language and foundational concepts like Data Structures & Algorithms (DSA) significantly impacts the efficiency, reliability, and scalability of applications.

**Java: A Foundation of Robustness and Portability:** Java has emerged as a preferred programming language for a myriad of applications, ranging from desktop to mobile and web development. The reasons for choosing Java as the foundation for our program are manifold:

1. **Platform Independence:** Java's "Write Once, Run Anywhere" principle makes it highly portable

across various platforms. The Java Virtual Machine (JVM) allows compiled Java code to be executed on any system, irrespective of its underlying architecture, thus ensuring our program's compatibility and reach.

1. **Strong Type System:** Java enforces a strong type system, minimizing the occurrence of runtime

errors. This results in more robust and reliable code, reducing the likelihood of unexpected crashes or vulnerabilities.

1. **Memory Management:** Java's automatic memory management through garbage collection

relieves developers from manual memory allocation and deallocation. This mitigates the risks associated with memory leaks and enhances program stability.

1. **Vast Standard Library:** Java's extensive standard library provides pre-built classes and methods

for various tasks, saving development time and effort. This empowers developers to focus on program-specific logic rather than reinventing the wheel.

1. **Community Support:** Java boasts a massive and active developer community. This translates to

a wealth of tutorials, documentation, and online forums where solutions to common problems can be readily found.

**Data Structures & Algorithms: The Backbone of Efficient Computing:** The selection to emphasize Data Structures & Algorithms (DSA) in our program design is grounded in the understanding that efficient algorithms and optimized data storage play a pivotal role in software performance. Here's why DSA is a paramount consideration:

1. **Optimized Resource Utilization:** By implementing appropriate data structures and algorithms,

we can ensure that our program utilizes system resources efficiently, leading to reduced execution time and improved scalability.

1. **Problem-Solving Efficiency:** Proficiency in DSA equips developers with the ability to devise

elegant solutions for complex problems. An in-depth understanding of algorithms allows us to choose the most suitable approach for various scenarios.

1. **Coding Interviews and Industry Demand:** DSA knowledge is a common criterion in technical

interviews and evaluations for software engineering roles. Therefore, focusing on DSA enhances not only the quality of our program but also our employability in the software industry.

1. **Foundation for Advanced Topics:** DSA serves as a foundation for advanced topics like

machine learning, artificial intelligence, and cryptography. By mastering these fundamentals, we set the stage for future growth and learning.

In conclusion, the decision to opt for Java as the programming language and to emphasize Data Structures & Algorithms in our program is driven by the desire to create reliable, efficient, and future-proof software. Java's portability, strong typing, memory management, and extensive library complement our goal of building a robust application. Simultaneously, a solid understanding of DSA ensures that our program's performance is optimized, and our problem-solving skills are honed. This strategic choice is poised to lay the groundwork for a successful and impactful software endeavor.

1. **Profile of the Problem: Algorithm Simulator**

The "Algorithm Simulator Project" is designed to address a fundamental challenge faced by learners and developers in the field of computer science: comprehending the intricate workings of various algorithms and their real-world applications. This section sheds light on the problem the project seeks to solve and the educational gaps it aims to bridge.

**Identification of the Problem:** Algorithms serve as the backbone of modern software systems, influencing efficiency, performance, and user experience. However, grasping the nuances of diverse algorithms and their practical implementations can be a daunting task for learners, particularly those who are new to computer science. As algorithms grow in complexity, understanding their step-by-step operations, optimizations, and trade-offs becomes increasingly challenging.

**Relevance and Impact:** The significance of algorithmic understanding transcends theoretical knowledge—it directly influences the quality of software development and problem-solving capabilities. Proficiency in algorithms is essential for crafting efficient code, optimizing processes, and tackling real-world challenges in fields like data analysis, artificial intelligence, and network optimization.

**Learning Gaps:** Learners often struggle with visualizing how algorithms function, leading to a gap between theory and practical application. Many learners find it difficult to grasp concepts like recursion, sorting algorithms, and graph traversal due to the abstract nature of these topics.

**Educational Necessity:** An algorithm simulator addresses the educational necessity of providing learners with a tangible, visual representation of algorithm execution. By offering interactive and dynamic simulations, learners can observe algorithms in action, understand their behavior, and experiment with different inputs, fostering a deeper understanding of their functionality.

**Expected Outcomes:** Upon completing the Algorithm Simulator Project, learners can expect heightened algorithmic proficiency, improved problem-solving skills, and a stronger foundation for tackling complex programming challenges. Additionally, learners will be better prepared for technical interviews and real-world software development tasks.

1. **Problem Analysis**

* **Product Definition: Algorithm Simulator:** The Algorithm Simulator is a software application

designed to visualize, simulate, and evaluate the behavior and performance of various algorithms across different input scenarios. It provides a user-friendly interface that allows users to input data, select algorithms, and observe their execution step by step, enabling a better understanding of algorithmic concepts and aiding in algorithm selection for specific tasks.

* **Feasibility Analysis:** Before diving into a project, it's important to assess its feasibility in terms

of technical, operational, and financial aspects. Here's a feasibility analysis for project:

1. **Technical Feasibility:**

* **Assessment:** Is the required technology and expertise available to develop the simulator?
* **Consideration:** The project might require advanced programming skills, knowledge of algorithm

design, and experience in creating interactive visualizations.

* **Conclusion**: If the development team possesses the required technical skills and tools, technical

feasibility is favorable.

1. **Operational Feasibility:**

* **Assessment:** Will the simulator be able to meet user requirements and expectations?
* **Consideration:** The simulator should offer a user-friendly interface, accurate algorithm

representation, and effective visualization tools.

* **Conclusion:** Operational feasibility depends on how well the simulator meets user needs and

how easily users can interact with it.

1. **Time Feasibility:**

* **Assessment:** Can the project be completed within the desired time frame?
* **Consideration:** Developing a comprehensive simulator with accurate algorithm

implementations and user-friendly features can be time-consuming.

* **Conclusion:** Time feasibility depends on the complexity of the simulator and the development

team's capacity.

1. **Implementation**

The project is implemented using Java, where we've developed a terminal-based algorithm simulator aimed at visualizing the behavior and execution of various algorithms. This simulator offers a platform for users to interactively explore algorithmic concepts through a command-line interface. The primary focus is on providing a clear step-by-step visualization of algorithmic processes, enabling users to gain a deeper understanding of their functionality and performance. Through this approach, we aim to bridge the gap between theoretical knowledge and practical insights in the realm of algorithms.

**The Algorithm Simulator visualize the following Algorithms:**

1. **Sorting Algorithms:**

* Bubble Sort
* Modified -Bubble Sort
* Selection Sort
* Insertion Sort
* Merge Sort

1. **Arrays Operations**

* Insertion (At beginning, at Kth position, at end)
* Deletion (At beginning, at Kth position, at end)
* Reversing Array

1. **Stack Operations**
2. **Queue Operations**
3. **Circular-Queue Operations**

Project Git-Hub Link: <https://github.com/Shreyash-Pandey90/Algorithm-Simulator>

**Here are some screenshots showcasing the results of the project**:

1. Asking the user to select the type of operation want to perform 🡪

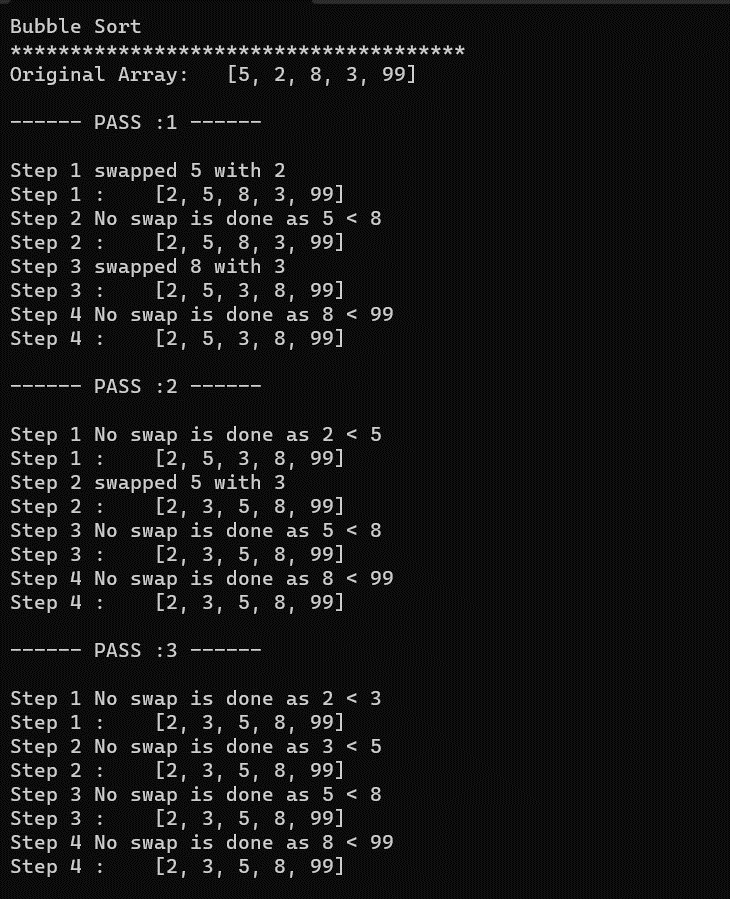
A screenshot of a computer program

Description automatically generated

**Fig 6.1 Code Output**

2. We can select any of the algorithm and give input to it then it will visualize my algorithm🡪

* Here we have selected the sorting algorithm.
* Then we will give input and then we will select the type of sorting🡪



**Fig 6.2 Code Output**



**Fig 6.3 Code Output**

**Array is sorted.**

**7. Learning Outcome from Summer Training at CipherSchools**

During my summer term training, I gained a comprehensive understanding of Java programming and foundational Data Structures and Algorithms (DSA) concepts through a structured curriculum provided by Cipher Schools. Over the course of the training, I covered a wide range of topics that equipped me with valuable skills and knowledge. Here are the key learning outcomes from my training:

1. **Java Programming Proficiency:** I acquired proficiency in Java programming, starting with the

fundamentals such as data types, operators, and object-oriented concepts. Through lectures and practical exercises, I learned how to work with objects, classes, constructors, and static variables. This foundation has enabled me to write Java programs effectively.

1. **Data Structures and Algorithms:** The training extensively covered various data structures and

algorithms. I learned about arrays, linked lists, stacks, queues, trees, and graphs. I also delved into algorithms for sorting, searching, and dynamic programming. This knowledge has provided me with a solid foundation for solving algorithmic problems.

1. **Problem-Solving Skills:** The training placed a strong emphasis on problem-solving techniques.

I practiced solving algorithmic problems, enhancing my ability to analyze problems, design efficient algorithms, and implement effective solutions. I learned to consider factors like time and space complexity while approaching different problems.

1. **Algorithmic Thinking:** Through lectures and coding exercises, I honed my algorithmic thinking

skills. I learned how to break down complex problems into smaller, manageable steps and develop algorithms that provide optimal solutions. This skill is essential for tackling various challenges in programming and computer science.

1. **Data Structure Implementation:** I gained hands-on experience in implementing various data

structures from scratch. This practical exposure helped me understand the internal workings of these structures, such as linked list insertion, tree traversal, and graph representation.

1. **Understanding Software Complexity:** I learned about time and space complexity analysis,

which is essential for evaluating the efficiency of algorithms. This understanding enables me to choose appropriate algorithms for different scenarios.

1. **Graph Algorithms and DFS:** I gained insight into graph representation and traversal techniques,

focusing on Depth First Search (DFS). This knowledge is valuable for solving problems related to network analysis and traversal.

1. **Algorithmic Problem Solving:** I practiced solving a variety of algorithmic problems, ranging

from basic to advanced. This experience has enhanced my problem-solving skills and confidence in tackling coding challenges.

Overall, my summer term training at Cipher Schools has equipped me with a solid foundation in Java programming, data structures, and algorithmic problem-solving. The exposure to diverse topics and hands-on exercises has prepared me to apply these skills effectively in future programming endeavors.

**8. Gantt Chart**

Outlined below is a Gantt chart capturing the chronological progression of topics covered and achievements throughout the summer training at Cipher Schools.

A graph with different colored rectangles

Description automatically generated

1. **Project Legacy**

The Algorithm Simulator project leaves a legacy by empowering learners and enthusiasts to demystify complex algorithms with ease. It stands as a tool that bridges the gap between theoretical knowledge and practical understanding, fostering a deep comprehension of algorithmic processes. This legacy extends through its user-friendly interface, enabling individuals to visualize, experiment, and master algorithms, contributing to a community of skilled programmers and problem solvers.

* **Technical Lessons Learned:**

**Algorithm Abstraction:** The project reinforced the importance of designing algorithms with modularity and abstraction in mind. Creating reusable and encapsulated algorithm components allowed for easier integration and maintenance.

**Real-time Visualization:** Implementing real-time algorithm visualization taught the significance of balancing efficiency and responsiveness. The visual components required optimization to provide a smooth user experience.

**Data Structure Implementation:** Developing the simulator involved practical implementation of various data structures. This hands-on experience deepened understanding of how these structures function internally.

* **Marginal Lessons Learned:**

**Problem Solving:** The intricacies of designing the simulator deepened problem-solving skills. Identifying algorithmic bottlenecks, optimizing code, and addressing unexpected challenges were valuable lessons.

**Documentation:** Creating comprehensive documentation for the simulator demonstrated the importance of clear explanations and instructions. This skill is essential for aiding users and maintaining the project in the long run.

These technical and marginal lessons contribute to a holistic learning experience, encompassing both technical skills and broader project management and communication capabilities.

1. **Bibliography**

* <https://www.javatpoint.com/java-tutorial>
* <https://www.geeksforgeeks.org/>
* <https://venngage.com/>
* <https://www.cipherschools.com/courses>
* <https://towardsdatascience.com/>
* <https://www.thecodehelp.in/>